Advanced Air Transportation Concepts and Technologies (AATT)

RTO-80: Development of a Pre- and Post-Processing Environment for ACES that Incorporates JView Software for Visualization

Deliverable Item 4:

RTO80 Final Report

30 September 2004

Contract Number NAS2-98002

Prepared for:
NASA Ames Research Center

Prepared By:
Science Applications International Corporation,
Simulation and Information Technology Operation,
Arlington, VA

This page intentionally left blank.

Table of Contents

1 Introduction

1.1 Background

This report summarizes research performed to develop and prototype a concept for pre-runtime tools for the Virtual Modeling and Simulation (VAMS) Airspace Concept Evaluation System (ACES) simulation system. The research had the additional goal of evaluating the Air Force Research Labs JView Application Programmer's Interface (API) software for use in two- and three-dimensional visualization of NAS data

The overall goal of the VAMS Project is to provide the foundations required to define and assess the next generation air transportation system. The VAMS Project is focused on identifying and assessing the performance of new operational concepts that, when incorporated into a future Air Traffic Management system, will result in a revolutionary improvement in system capacity, at an affordable cost and with no reduction in safety. In particular, the Virtual Airspace Simulation Technologies (VAST) Sub-Project seeks to develop airspace simulation environments with the capabilities to assess the integrated behavior of current and future air transportation system concepts and technologies at the system-wide level and at the detailed human-in-the-loop level. There are two research-areas associated with the VAST Sub-Project: Airspace Modeling and Simulation, and Real-Time Simulation. The work performed as part of this RTO supported the first research-area, which focuses on non-real-time modeling and simulation.

This RTO investigated the ability to create a pre- and post-processing environment for ACES. The products of the RTO work include engineering designs, software designs and prototype implementations of databases and tools.

1.2 ACES Architecture

From the outset the architectural design of ACES has encompassed both runtime and non-runtime applications. The ACES System/Subsystem Design Description (SSDD) / Software Design Document (SDD) [ACES CTOD 7.27/7.38], describe "a number of non-runtime components, … These components focus on scenario generation, configuration of simulation applications from ACES toolbox components, data management and assessment tools for analyzing simulation results." Figure 1, reproduced from the ACES SSDD, shows a set of non-runtime Simulation Configuration, library and assessment applications and as a persistent database holding ACES data.

The ATMSDI CTO-7 effort has developed substantial implementations of the runtime (yellow/orange in Figure 1) components of the ACES system, however the mainline ACES development effort has applied relatively little focus to the non-runtime (purple/white in the figure) components of the system.

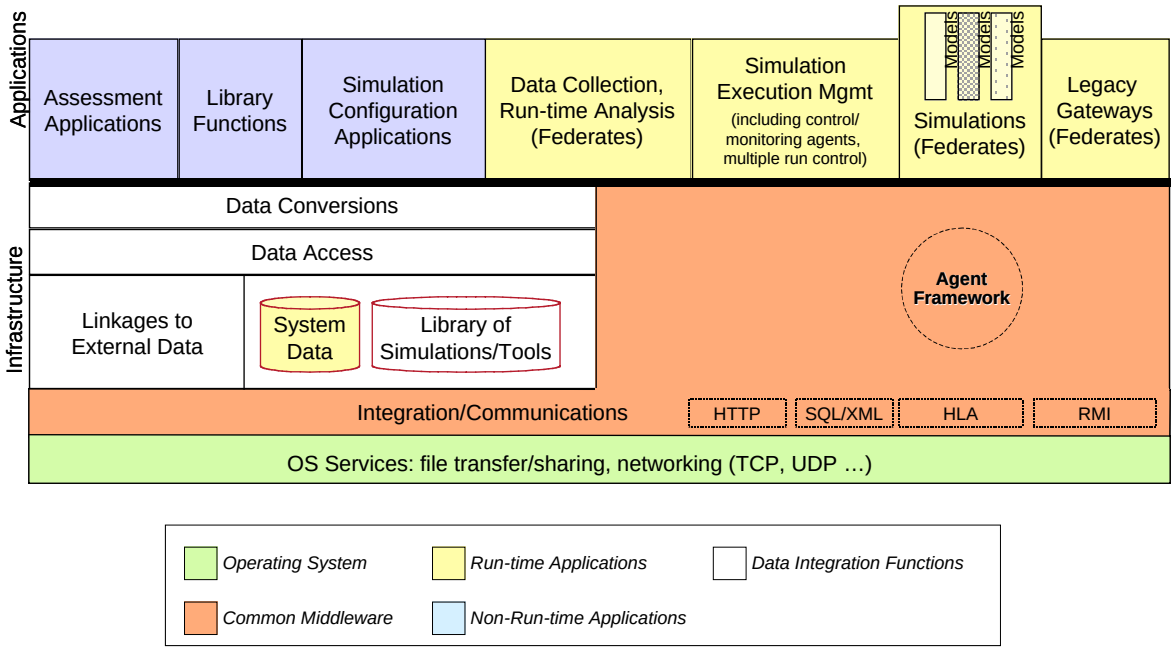

Figure 1. ACES Simulation System Architectural Approach

Figure 2 shows a process flow view of ACES. ACES is a high fidelity simulation of a complex system and as such makes use of a great deal of input data. ACES input data includes static data such as descriptions of the National Airspace System (e.g., sector boundaries), its hierarchy of control (e.g., sectors composing an ARTCC), and its users (that is, airlines and aircraft characteristics). Such data are typically constant over a large set of runs. ACES also contains data that vary to represent a particular day in the NAS. These data, which include items such as airline schedules, weather and the like, typically vary from run to run in an experiment. ACES data also includes system parametric data that are used to create excursions. Examples of this latter category include variations in sector and airport capacities.

The left-hand section of Figure 2 depicts a notion for the creation of ACES data. It shows libraries of ACES data and models that are composed to create the set of inputs required for runs. The middle section of the figure depicts execution using the composed inputs using ACES, and the right-hand section shows post-run analysis of ACES outputs.

Input data for baseline ACES releases primarily take the form of text files. Such “flat files” can easily be edited using a plain text editor, however this format has a number of drawbacks. First, as the collection of data grows, maintaining the set of files becomes difficult. For example, a modeling change in Build 3 required the addition of a data column to Flight Data Set files. A change such as this requires manual modification of a large number of files or maintenance of the ability to support an ever-growing number of legacy formats. Referential integrity

is difficult to maintain in flat files as well. As an example, adding an airport to ACES requires separate changes to a number of files, a time-consuming process. Last, using flat files it is difficult to characterize or perform quality checks on data. A question such as “how many flights in this Flight Data Set have Denver as their departure airport?” or “does this Flight Data Set have any flights serving airports that are not in my list of airports?” are virtually impossible to answer using flat files. Manual edits to flat files also tend to be error-prone, introducing data and formatting errors into the data.

In recognition of the shortcomings of flat files, it has generally been accepted that ACES input data will ultimately migrate from flat files to databases.

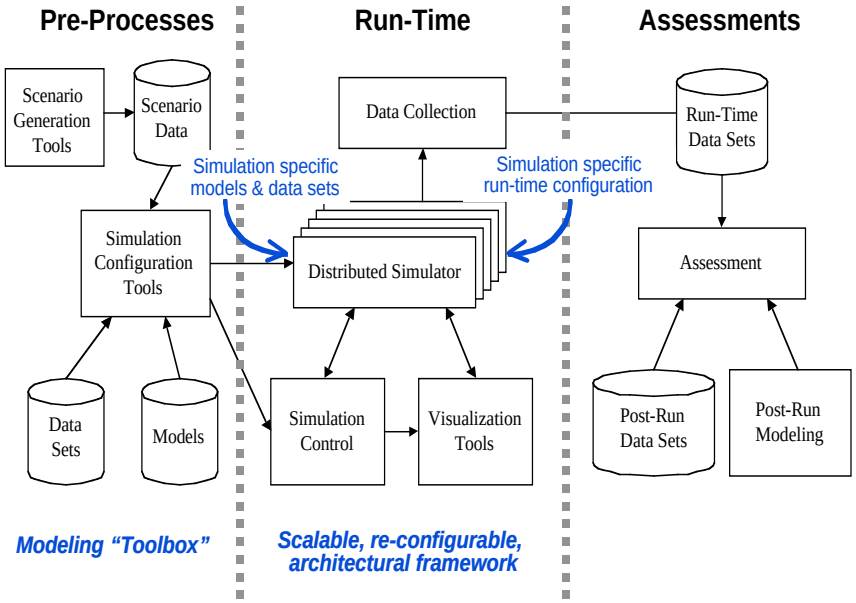

Figure 2: ACES Process Flow

Databases, though, are not a panacea in and of themselves. Storage of data in a relational database management system solves all of the flat file problems noted above, however database introduce two types of challenges of their own. First, efficient database design is more difficult than simply putting data in files. Second, while relational database management systems are powerful, structured tools, their native interfaces (typically SQL command lines or other query-based tools) are unfamiliar to analysts and so would present something of a learning curve to ACES users.

1.3 Project Goals

A primary goal of the RTO80 project effort, as stated in the RFP was “enhancing ACES usability by designing and developing pre-processing capabilities.” This goal was addressed through design and prototype implementation of a system

concept called the Scenario Processing and Data Environment for Simulations (SPADES).

The RTO80 Task Statement defined four major tasks to be performed under the effort, including:

1. **Separate the ACES Simulation Control and VST functions**: Within ACES, the Simulation Control and VST functions are not cleanly separated. Simulation Control and visualization are combined in a single application.
2. **Define the pre-processing environment**: Create a concept for a pre-processing environment for ACES.
3. **Prototype pre-processing software**: Create prototypes to demonstrate the system concept developed under item #2.
4. **Demonstrate Visualization of ACES Data Using Jview**: Investigate the applicability of the JView 3D API in visualizing ACES runtime and non-runtime data.

Effort against the first task listed above focused on creating a separate visualization federate using JView. In recognition of the relational database characteristics mentioned in the previous section, the effort on the second and third of these tasks focused on defining a database schema for ACES data, developing a set of prototypes of JView-based ACES tool capabilities and demonstration of how a data repository can be linked to existing ACES code through legacy flat file formats. The fourth task was accomplished through the use of JView in the prototype pre-processing tools and visualization federate.

1.4 Deliverables

1.4.1 Overview of Major Deliverables

The products of this research effort include designs, prototype implementations and documentation. The designs include database, engineering and software designs. The prototype software implementations include both software and associated user manuals describing the installation, configuration and use of the software, and images of the use of JView. The user manuals were written with target audiences of both NASA researchers who are experts with the ACES simulation system and concept developers supporting the VAMS Project who are not expert users.

The following list summarizes the project deliverables. Each major product (save for this report) is described in greater detail in subsequent sections.

SPADES Database

This deliverable was a representation of the Build 2.0.3 ACES data in a normalized relational database format. It was created to satisfy the requirements to “define the pre-processing environment” and “prototype pre-processing software”. Deliveries included:

- Database schema design that captured the breadth of ACES input data. Provided as a database design document
- Prototype implementation using the MySQL RDBMS, populated with example data drawn from the Build 2.0.3 ACES data set.

ASRTV Runtime Visualization

This deliverable was created to satisfy the requirement to "separate the ACES Simulation Control and VST". It consists of a separate runtime visualization federate that runs as part of an ACES federation. It displays a superset of the data displayed by VST and offers both two dimensional and three dimensional views. The deliverable also included ACES VST and Simstartup modifications required to integrate the additional viewer. Deliveries included:

- Initial software delivery compliant with ACES Build 2.0.3
- Updated version compliant with ACES Build 3.0.1 and incorporating JView fixes
- Engineering Design Document
- Software Design Document
- User Guide

Weather Editor / FDS Viewer

This deliverable, along with the database design, completed the tasks to "define the pre-processing environment" and "prototype pre-processing software". It demonstrates several capabilities. First, it demonstrates using JView to visualize ACES input data, in particular Flight Data Set trajectories and weather data. Second, it demonstrates a Simulation Configuration application that uses a graphical user interface to allow analysts to easily create large ACES data sets. Third, it shows how export utilities can be used to create legacy ACES data formats from database data. This intermediate capability that allows tools to be used to create data for ACES in advance of an adaptation of ACES code to read data directly from databases. Deliveries included:

- Software compliant with ACES Build 3.0.1
- Engineering Design Document
- Software Design Document
- User Guide

Final Report and Briefing

The present document is the final report. It summarizes the research performed under the task. It also includes an evaluation of JView as a basis for a replacement for the existing ACES visualization tool known as the VST.

Deliverable List

The table below summarizes the complete set of deliverables provided under this task.

Deliverable	#	Comment
Task Plan	1a	NA
Kickoff Meeting at NASA Ames	1b	NA
Monthly Status Report, including technical progress during the reporting month, plans for the following month, % complete by sub-task, risk items and mitigation plans[2]	2	NA
Software and Data required under Task 2/Sub-tasks 1 and 2 and Task 4[3]	3a	Included SPADES Database Schema, initial release of ASRTV viewer (compatible w/ ACES Build 2.0.3), updated release of ASRTV viewer (compatible w/ ACES Build 3.0.1). The Schema included: ■ Schema Design Document ■ Example MySQL database ■ Database utilities (FDS data loader, FDS/Sector Intersection Finder) The ASRTV Viewer Included: ■ Software, including ACES modifications ■ Engineering Design Document ■ Software Design Document ■ User Guide
Software and Data required under Task 2/Sub-task 3 and Task 3[3]	3b	Included SPADES Weather Editor/FDS Viewer. The ASRTV Viewer Included: ■ Software ■ Engineering Design Document ■ Software Design Document ■ User Guide
Final Report	4	NA
Final Oral Briefing	5	NA

2 SPADES Database

2.1 Description

The SPADES database provides database storage for the majority of ACES input data, including VST Configuration, Simulation Configuration, Static Input Data, and Flight Data Set Data. Certain classes of externally generated data, in particular RUC wind data and BADA aircraft data are not included in the SPADES design. The decision not to include these items was made based on the fact that these are static data that are used "as is" as provided from their sources.

The database schema takes as its basis the set of simulation data defined for ACES Build 2.0.3; however it extends the ACES data in several directions. SPADES adds the concept of user experiments to the current ACES data set. An experiment is defined as a particular instance of a simulation scenario that is run at a particular time. The Experiment concept ties together the various sets of data associated with an ACES run, including Simstartup data, Local Data Collection (LDC) data, data associated with VST configuration and other related data items. It is designed with the notion that in a shared development environment experiments could be created, stored, shared, copied and reused by ACES users.

The SPADES database has been designed so that it could be used directly by ACES if ACES were to be modified to read database input files. It can also be used as a basis for generating data in existing ACES data file formats. This latter capability is demonstrated in the Weather Editor.

The data remain tied to the source ACES structures but in key areas have been normalized for better data reuse across simulation experiments. SPADES provides a normalized view of Flight Data Sets that separates routes from specific flight plans, and allows for hierarchical creation of flight data sets. A user can take existing flight data sets and combine them to build new flight data sets for their experiments. The separation of routes from flight plans also facilitates reuse and preprocessing, as a user can quickly create a new flight data set by selecting 'off the shelf' flight plans, and further select from a variety of 'off the shelf' predefined routes that are unique between airports or pass through particular sectors.

The database schema also augments the current ACES data with some concepts specific to SPADES. The primary example of this is that the SPADES database stores weather system data for the Weather Editor. The database allows for storing a shared set of weather systems that a user can select, customize, and reuse in their experiment.

The products of the SPADES database design include a design document that defines the SPADES Schema and Naming Conventions and describes the design logic behind the design. The design document describes the contents of the database. The document both links the tables back to the Build 2.0.3 source data

an, for major data groups describes how data files can be generated from ACES tables. The schema document does not attempt to re-describe the contents and semantics of ACES data as these topics are already covered in the ACES User Manual (CTO7 CTOD7.29).

2.2 Development Process

The database design began with identification of the set of ACES input data to be included in the database design. This included primarily the data found in:

%CTO7_HOME%/Build1/modules/cto7sim/data

%CTO7_HOME%/Build1/modules/simstartup/bin

and the Local Data Collection data found in

%CTO7_HOME%/Build1/modules/Cybele

Other data groups were specifically excluded from the design. In some cases this was because they were considered to be static data that were used as-is from external sources (e.g., Rapid Update Cycle [RUC] wind data and BADA aircraft characteristics). In other cases the data were considered to be associated with a lower level middleware tool and changed only infrequently, typically only with changes in the ACES software (e.g., Federation Object Model [FOM] data). Cybele configuration data was deemed to belong to this third category, though users may in fact modify Cybele configuration parameters to control system features such as logging and profiling. It is recommended that a full implementation of the SPADES database include these dynamic Cybele parameters as well.

The philosophy of the database design was to preserve wherever possible the existing structure of the data. This will ultimately make it easier to integrate the database schema with ACES software. Balancing this, however, was the fact that there was room for improvement as the existing set of ACES data is disjointed and contains some formatting inconsistencies. There were also opportunities to improve the organization of the data via data normalization. Thus, the resultant design is easily recognizable to anyone familiar with the ACES but also contains new concepts and organization to better integrate the total data set.

As described above, the schema design adds the unifying concept of an Experiment, which ties together the disparate ACES data groups (LDC data, model data, scenario data, etc.). Experiments also allow experiment-specific values to override systems defaults for certain data values. The design process also noted a number of other common items that appear across multiple groups of ACES data. An example of this case is aircraft type (which appears in Flight Data Sets and Aircraft Transit Times). Care was taken such that these items were named and represented consistently across the tables of the database.

Last, the analysis revealed that ACES data clustered into six natural groupings as follows:

- Experiment Specific Data: These data capture the definition of an experiment and any overrides of default data that are specific to the experiment. An experiment definition includes selection of Flight Data Set, LDC configuration, scenario event set and other parameters. Overrides would include, for example, customized values for airport TFM operational parameters.
- Configuration Data: These data define the execution parameters for the simulation, that is, number of Generic Masters, assignment of federates to hosts, etc.
- Data Collection Data: These items specify the configuration for Local Data Collection for a particular experiment.
- Reusable Experiment Data: These data are part of the representation of a particular "day in the NAS" but could be reused across experiments. Examples here include Flight Data Sets that may be run under differing NAS capacities or Weather Days that may be used against various demand sets.
- Static Lookup Tables: These represent baseline data that defines the NAS, such as center boundaries and the set of airports in the NAS.
- Editor Data: these represent data that is specific to SPADES tools rather than to ACES or the NAS.

Based upon this analysis and categorization a database design was developed. An overview of the database schema is shown in Figure 3. While this highly condensed overview of the schema is not particularly legible it does serve to give an idea of the size and complexity of the schema. More detail is provided in the schema design document.

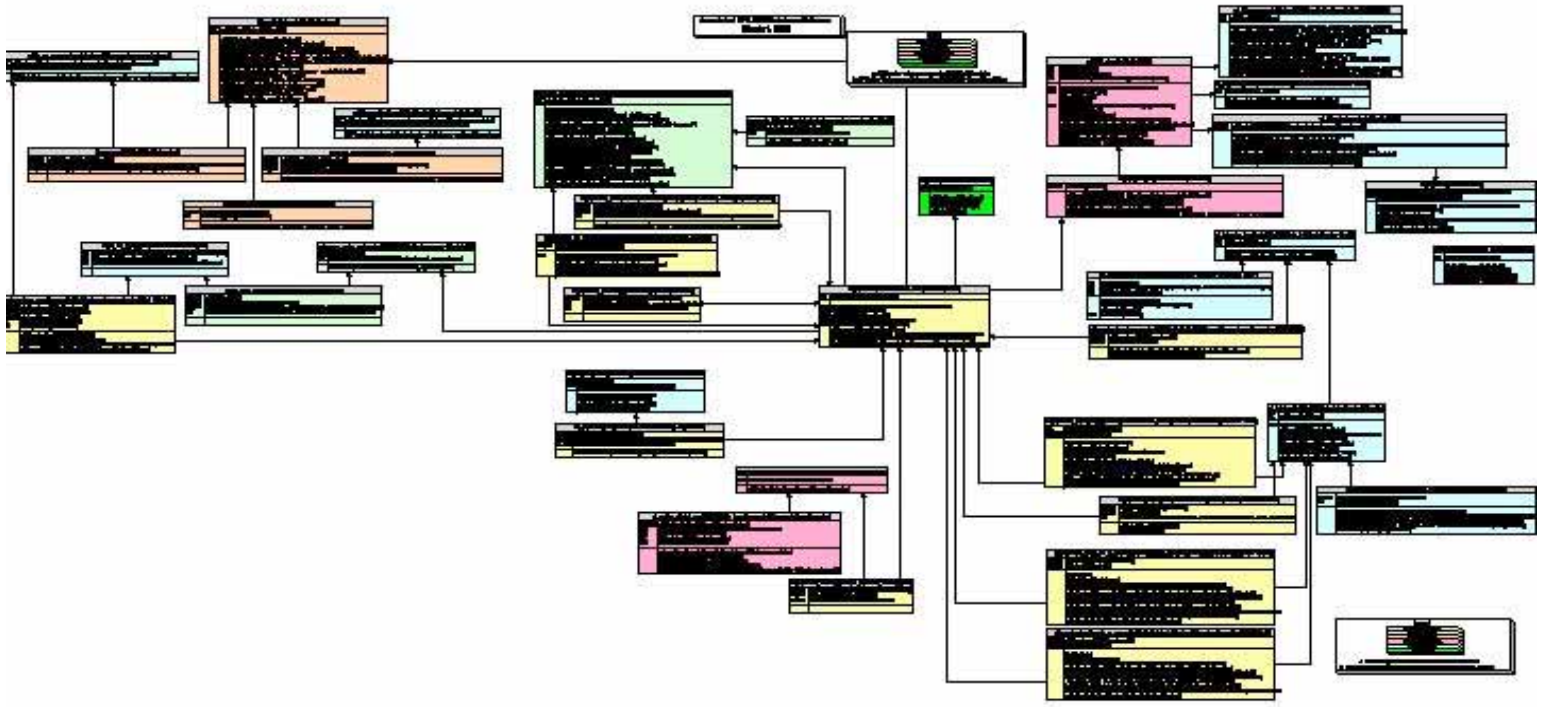

Figure 3: SPADES Database Schema Overview

2.3 Implementation

The design described in this document has been implemented and populated with sample data using the MySQL relational database management system. The prototype implementation comprises thirty nine tables, all of which are populated with example data from ACES and/or SPADES.

3 ASRTV Viewer

3.1 Description

The ACES-SPADES Runtime Viewer (ASRTV) is a tool for visualizing simulation data from the Airspace Concept Evaluation System (ACES) during runtime. The program provides a map-based display of ACES data as the simulation executes, including intent, trajectories and flight events. The tool's point-and-click interface allows users to drill down and view more detailed textual information for many ACES simulation objects.

The tool is implemented as a separate Cybele-HLA based federate that runs within the ACES federation. Implementing the viewer this way decouples it from the control functions of the existing VST visualization tool. An ACES federation can have multiple Runtime Viewers displaying different parts of the NAS. The Runtime Viewer can also be driven by other HLA-based data sources that comply with the ACES FOM. For example, the viewer could visualize fast-time data published through the RTI by a playback application based on Local Data

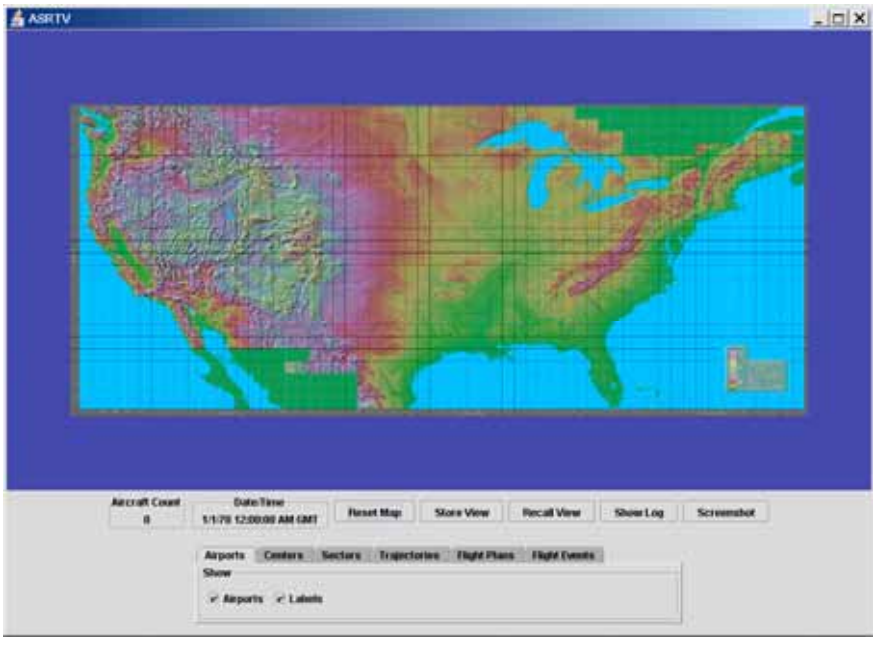

Figure 3: ASRTV Runtime Viewer Main Screen

Collection Data from an ACES run.

The driving requirements for the viewer come from the government's TO80 Statement of Work. These include the desire to "Separate the ACES Simulation Control and VST functions" and use "JView two-dimensional libraries to provide a better plan view map with more information available than in the current ACES visualization tool". The definition of a "better plan view map" was not defined in the SOW and was interpreted as follows: (a) preserve the display capabilities of VST; (b) implement existing visualization-related Software Change Requests (SCRs) to the extent possible; (c) capitalize on the capabilities of JView; (d) to the extent possible implement other features as described by NASA in Kickoff and Review meetings.

The following SCRs and NASA guidance were partially or fully implemented in the ASRTV Viewer:

SCR 28: The ASRTV viewer provides a graphical display of conflict events related to each flight. The conflicts are represented as icons along the flight's track. Additional information about a conflict can be viewed by selecting its graphical conflict icon with the mouse.

SCR26: By providing a 3D view, ASRTV allows viewing of flights' vertical trajectory profiles, as flown.

SCR40: ASRTV provides color-coded aging of flight trails.

SCR46: The ASRTV display is fully navigable using the mouse. All items shown on the 3D display are clickable.

SCR118: The display shows both intent and trajectories as actually flown.

SCR235: See discussion under SCR 28 above.

Figure 3a shows the major display features of the ASRTV viewer. Figure 5b shows an example of a 3D aircraft visualization in the ASRTV viewer.

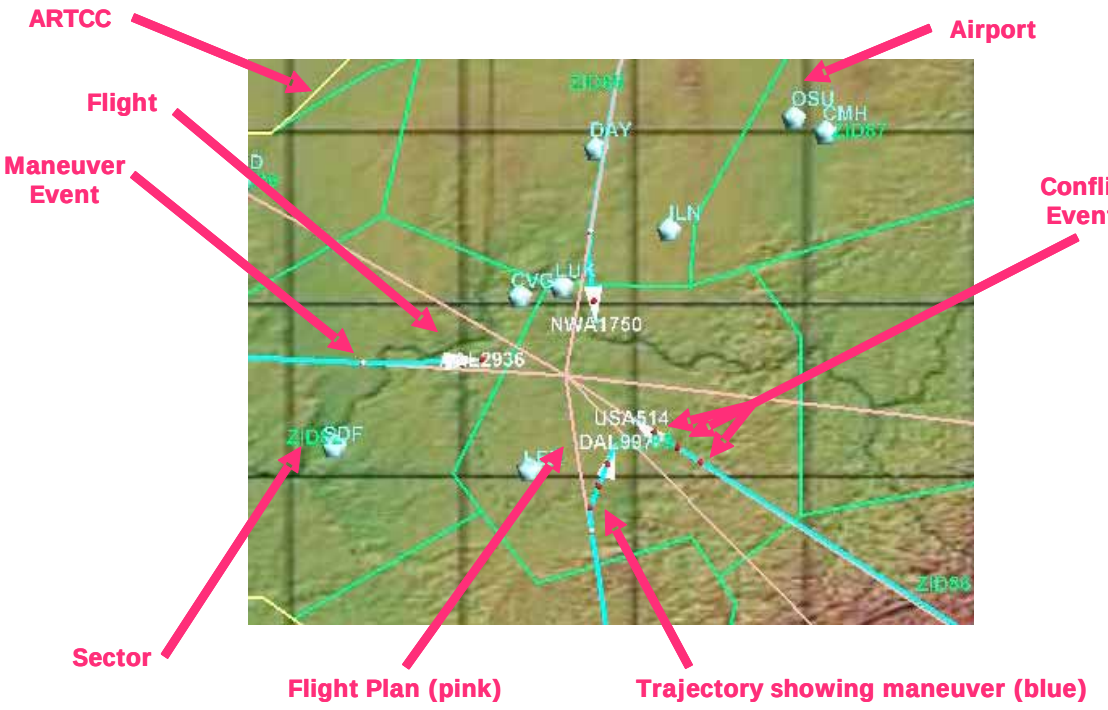

Figure 3a: Display Features of the ASRTV Viewer

Figure 5b: Display Showing 3D Modeling

3.2 Features

The major features of ASRTV include:

- Graphical Visualization of ACES data at runtime
  - Flight Plans
  - Tracks as flown, including aging of trails
  - Events: conflicts, maneuvers, boundary crossings, TOC, TOD
  - Static airspace data, including airports, ARTCC and sector boundaries
  - Simulation time and number of aircraft
  - Realistic background map
  - Ability to show aircraft as icons or using 3D models
  - Save and restore views (pan, zoom and center location)
- Drill-down
  - Selection of graphical items reveals more data (e.g., selecting a flight reveals its flight ID, location, speed, origin and destination, etc.)

- Mouse-based interface
    - Full pan, zoom and 3D rotation using the mouse
    - Selection of graphical objects using the mouse
    - GUI-based control of all features
- Screen capture and save to JPEG format
- Ability to use flight icons or full 3D models of aircraft
- Compatibility with HLA (RTI NG 1.3), ACES Build 2.0.3 and 3.0.1, ACES Simstartup

ASRTV has been tested up to a 4800 flight scenario. It should be noted that running the full JView 3D graphics engine is more computationally intensive than the simple 2D vector graphics of VST and hence for large scenarios the use of ASRTV can decrease ACES execution speed.

3.3 Development Process

The architecture of the ASRTV application closely resembles that of the existing VST tool. Development began by adding new Swing and JView classes to the VST code. The result was a tool that displayed two maps, the original 2D VST map and the new JView-based 3D map. This process ensured that the new visualization code would be called appropriately. Once this tool was tested for consistency between the two maps, the 2D and configuration-related code was removed leaving an independent federate.

The strength of this approach is that it ensured that the new federate was consistent with ACES VST and Generic Master (GM) software in the way it initializes, is configured and interacts with other federates in the system. There is one note associated with this, however, in that any other use of ASRTV, for example, for playback, would need to either mimic the ACES initialization approach (e.g., by playing back configuration data) or modify ASRTV to implement a different initialization approach.

3.4 Areas for Potential Future Enhancement

The ASRTV development effort was something of a proof of concept for, as stated in the RFP "the applicability of the JView 3D API in visualizing ACES runtime … data." While it provides a fully capable viewer, additional features were identified that could further improve the utility of this application. These include:

Item	**Discussion/Benefit**
1. Implement post-runtime playback capability.	This would provide the ability to play back LDC data to the viewer in very fast time.

	fast time. This feature would support post-run analysis in a multiple run environment.
2. Add weather overlays	Used in conjunction with the Weather Editor, this would display Weather Affected Areas on the Runtime Viewer to provide visual context for the impact of weather on simulation execution.
3. Full 3D visualization	Currently sectors and center are shown as 2D projections onto the map. The enhancement would display these regions as 3D translucent areas. This richer representation of the 3D airspace could enhance analytical understanding of ACES execution.
4. Convert background map to use JView V2.0 continuous level of detail (CLOD) display of Digital Terrain Elevation Data (DTED)	This would replace the current 2D map image with a 3D terrain representation. This would improve visual fidelity and would also eliminate the manual process of geo-registering map images.
5. 3D flight icons	Currently flight locations are displayed as an arrow or using a single 3D icon. This effort would match the 3D representation to the appropriate aircraft type for a flight. This would provide greater visual realism for demonstrations.
6. Terminal area display enhancements	This would augment the terminal area display to represent enhancements in ACES terminal area modeling.
7. Aggregate level displays	This would add aggregate information displays as prototyped by AFRL (such as total delay accrued and aircraft densities) as an overlay on the viewer display.

4 SPADES Weather Editor

4.1 Description

The Weather Editor is a data preparation and viewing tool for ACES. It was created to demonstrate the concept of data pre-processing tools for ACES and to demonstrate the use of the Air Force Research Laboratory JView software development kit within ACES.

4.1.1 Weather Editing Concept

ACES does not contain an explicit representation of convective weather. Rather, weather is represented implicitly by its impact on sector and airport capacities. Thus, modeling a moving weather cell within ACES requires time intensive data preparation. The analyst must manually determine the set of airports and sectors affected by the weather at each quarter hour simulation interval and must create data files to drive the simulation to implement these impacts. The data that must be created is a set of ACES *scenario events*. Scenario events allow users to (among other things) override default values based on time or events triggered by the evolution of the simulation scenario. Given the large number of sectors and airports modeled in ACES, manually generating scenario files for a large, moving weather system rapidly becomes impractical.

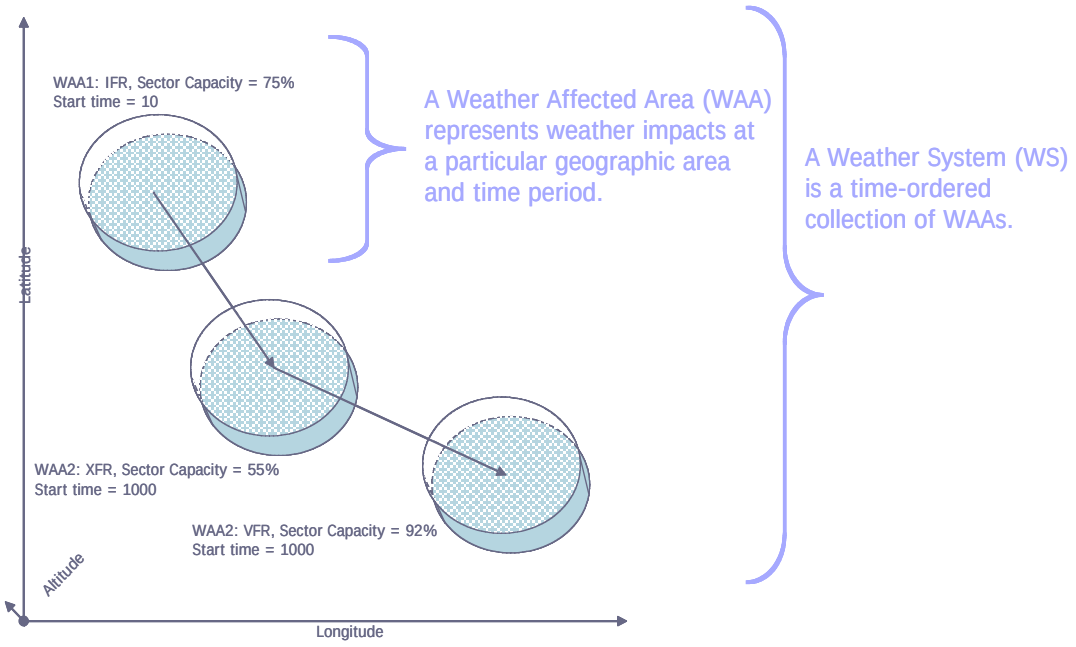

Figure 3: Weather Representation

SPADES introduces an explicit concept of weather that facilitates modeling of convective weather within ACES. The SPADES database and Weather Editor support the creation of *Weather Systems*. A Weather System is a container for a time series of *Weather Affected Areas (WAA)*. Each Weather Affected Area represents a 3D geographic region, a time period and a set of impact parameters. Using the Weather Editor the user can specify a weather system as a series of WAAs. The program then computes the intersections between these WAAs and NAS elements (airports and sectors) and generates appropriate ACES scenario data files. Thus the manual process of scenario file creation is replaced

by a graphical point-and-click process of drawing Weather Affected Areas on a Map.

The Weather Editor was developed to facilitate modeling of weather within ACES, however the tool can also be used to represent other airspace phenomena that can be modeled via capacity changes. An example of this would be modeling of a

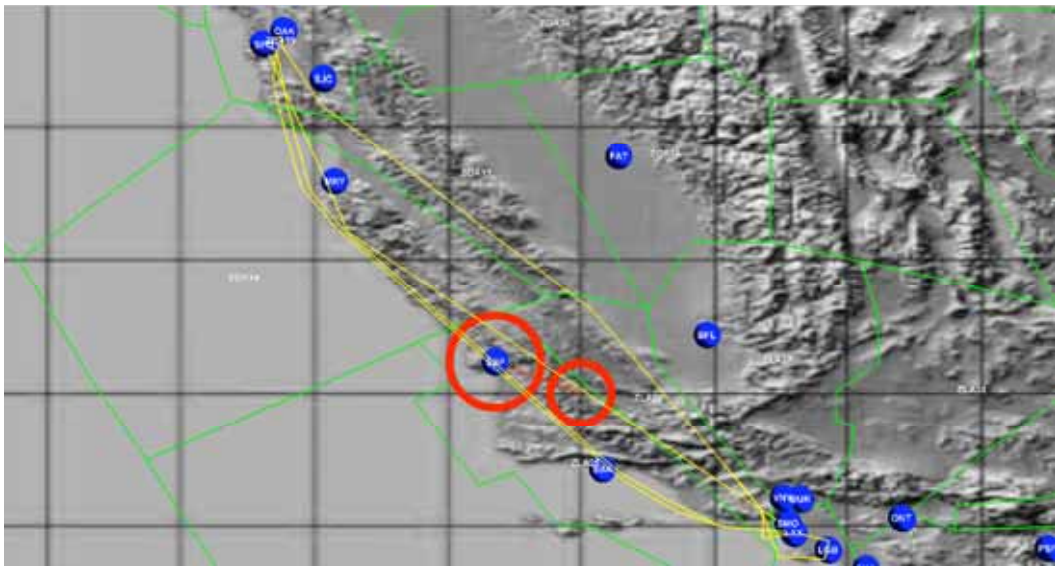

Figure 4*:* Example Weather System

Special Use Area that is activated for certain periods within the day.

Figure 3 diagrams the weather representation. Figure 4 shows an example of how the Weather Editor displays a weather system made up of two Weather Affected Areas. In the latter image, the WAAs are shown as the two red circles, airports are shown as blue spheres and sector boundaries are shown in green. The images also show a set of flight data set planned trajectories, shown in yellow. The ability to display FDS data is discussed in a subsequent section.

4.1.2 Weather Editing Process

This section presents a process-oriented view of creating weather with the SPADES Weather Editor. The given example shows how a user would model a line of storms moving across the SouthEast. The steps involved would be as follows:

1. Decide on the data that is to be modeled – for example, by sketching out a weather pattern or obtaining a set of weather photos. This is done outside of the Weather Editor.
2. Using the editor, create a weather system within the database. Add WAAs to the weather system to model the movement of the storm (see the

image below for an example). Using the mouse (for location/size) and the spreadsheet in the WAA List window (other parameters) enter data for each WAA.

3. As needed, add additional weather systems and WAAs to model the extent of the storms. Figure 5 shows an example of the Weather Editor display populated with weather systems. The figure shows a line of storms with a considerable North/South extent. This is modeled in the weather editor using three parallel weather systems. Note that it is possible to copy and paste entire weather systems to easily produce data sets such as that shown above.
4. Save the data set to the database.
5. Use the editor's Weather/Scenario Events/Generate function to populate the scenario event table with weather-related scenario events. From this table, generate an ACES format scenario event file.

4.1.3 Weather Generation Algorithms

As mentioned above, ACES does not at present contain an explicit weather agent capable of directly interpreting weather data. Thus, weather data must be translated into a form that can be used by ACES. In practice with Build 3.x, this

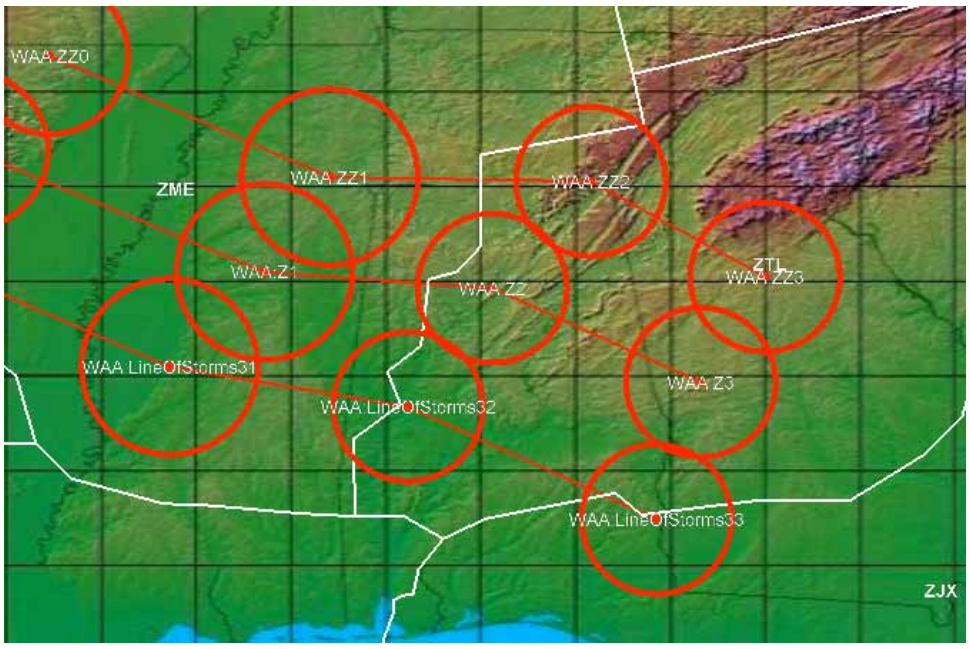

Figure 5: Example of drawing a moving line of storms in the Weather Editor

means translating weather data into a series of ACES scenario events. Generation of scenario events from weather system data is a three step process. First, the WAAs in a each weather system are interpolated to provide continuous

weather coverage. Second, the impacts for the sectors and airports are computed. Last, scenario events are generated and exported.

Interpolation

The need for interpolation comes about for two reasons. First, it is a convenience feature that reduces the number of WAAs the user has to define. Second, it reduces the possibility of missed impacts due to holes or scalloping at the WAA boundaries. For example, Figure 6 shows how an airport (EWR) could be missed because it is outside the boundaries of two defined WAAs even though it is clearly in the path of the weather. The figure shows how interpolation improves the sampling of the path of a weather system.

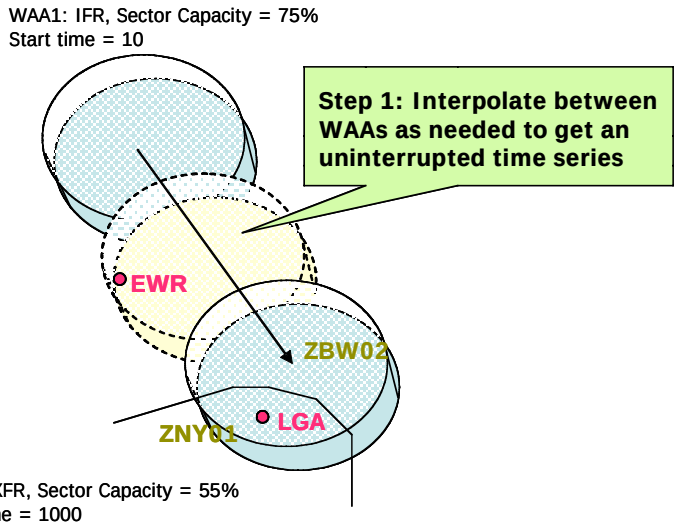

Figure 6: WAA Interpolation

ACES sector and airport capacities are specified for quarter hour intervals. The WAA interpolation algorithm interpolates between WAAs to 7.5 minute increments as a sort of Nyquist Rate – it does not completely eliminate problems with holes and scalloping but greatly reduces their magnitude.

For the purposes of interpolation, a WAA is considered to be in effect at its specified location from its start time until the end of its linger period (startTime + lingerTime). Between this time and the start time of the next WAA in the Weather System the WAA is linearly interpolated at 7.5 minute intervals. The interpolated values include location, radius, minimum and maximum altitudes and sector impact percentage. The airport state is a discrete value and so cannot be interpolated. Instead, the value of the WAA itself is used for all interpolated WAAs until the next specified WAA in the weather system.

Interpolated WAAs have zero linger time.

Impacts

After the set of interpolated WAAs is generated the next step is to determine which sectors and airports are affected. An airport is affected by the WAA if its

location (in three dimensions) falls within the WAA's coverage envelope. A sector is affected if it intersects the WAA's coverage at all. No adjustment is made for the percentage of overlap between the WAA and the sector.

Resolution Among WAAs

A user may specify multiple overlapping WAAs, either within the same weather system or across weather systems in an experiment. In addition, Interpolation may generate additional overlaps. Thus, the algorithm contains a mechanism for resolving cases where an airport or sector is simultaneously affected by multiple WAAs.

The resolution algorithm is simple: Within a Weather System, of the WAAs that affect a given sector or airport, the one with the latest start time is chosen. Across Weather Systems, from among the WAAs the algorithm takes the one that has the greatest impact. For sectors, this means the WAA with the lowest value for *percentSectorCapacityChange*. For airports, this involves a lookup to see which of the specified airport states has the lowest capacity. This resolution algorithm supports bad weather insets, for example, a moderately stormy area with a smaller, intensely stormy core. It does not, however, support the inverse – a toroidal area of impact with a calm core.

Weather Export Algorithm

ACES does not at present have a mechanism for reading scenario from a database and so a function was created to export the weather data to ACES format scenario data files. The weather export algorithm translates sector and airport impacts identified by the weather generation algorithm into ACES scenario events. Since the weather generation algorithm calculates and stores quarter hour impacts for the entire duration of the simulation, the export algorithm is largely a matter of exporting data records from the SPADES database to appropriately formatted ACES scenario *.csv* files.

4.1.4 Flight Data Set Viewing Concept

The ACES Weather Editor also provides a means to graphically display ACES Flight Data Sets. A Flight Data Set (FDS) file consists of a collection of individual flight entries, where each flight carries its identifier, departure time, planned trajectory, cruise altitude and other data. The FDS viewing concept within the Weather Editor allows users to visualize FDS data by plotting trajectories in 3D

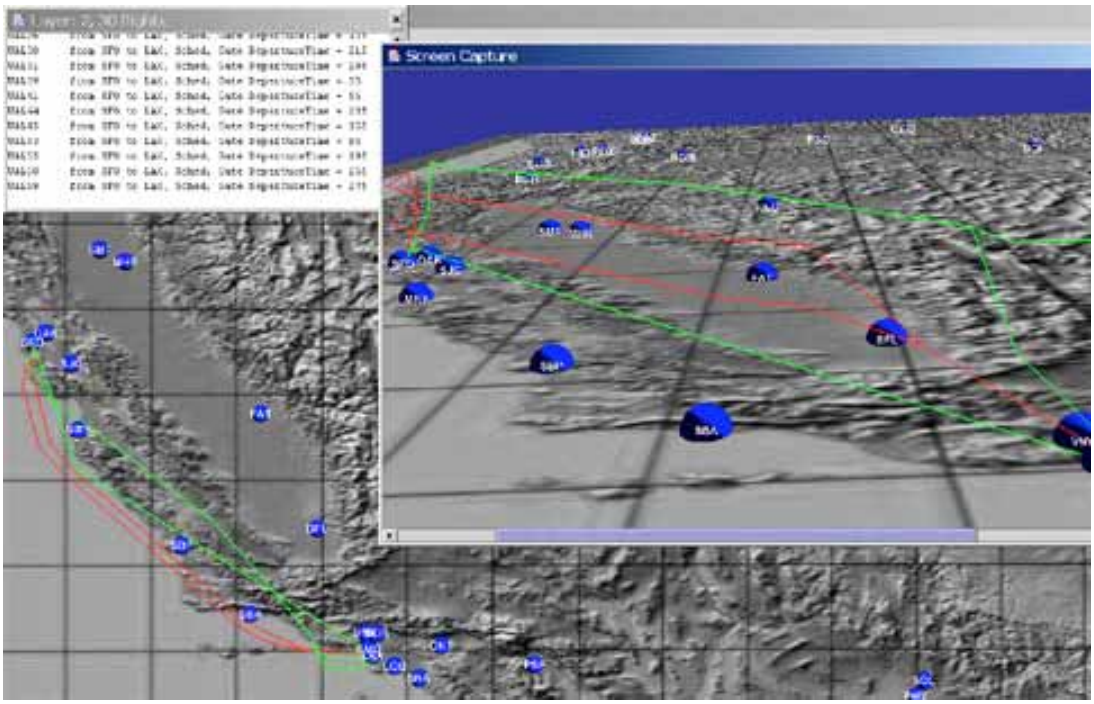

Figure 6: Flight Data Set View Showing Three Simultaneous Views: 2D, 3D and list of flights

relative to a map of the United States and NAS elements. FDS viewing allows the user to “slice and dice” Flight Data sets in various ways, allowing the user to limit flights displayed by time, origin and destination airports, flight ID and ARTCCs traversed. The trajectories displayed are simply the airport-to-airport fix lists contained in the input flight plan (FDS/ETMD) data files; they do not include any of ACES’ pre-flight computations such as insertion of meter fixes. Figure 6 shows three different views of a small Flight Data Set: 2D, 3D and a list of the flights. The flights are color-coded according to departure airport. Figure 6 shows an example of displaying a large flight data set in its entirety.

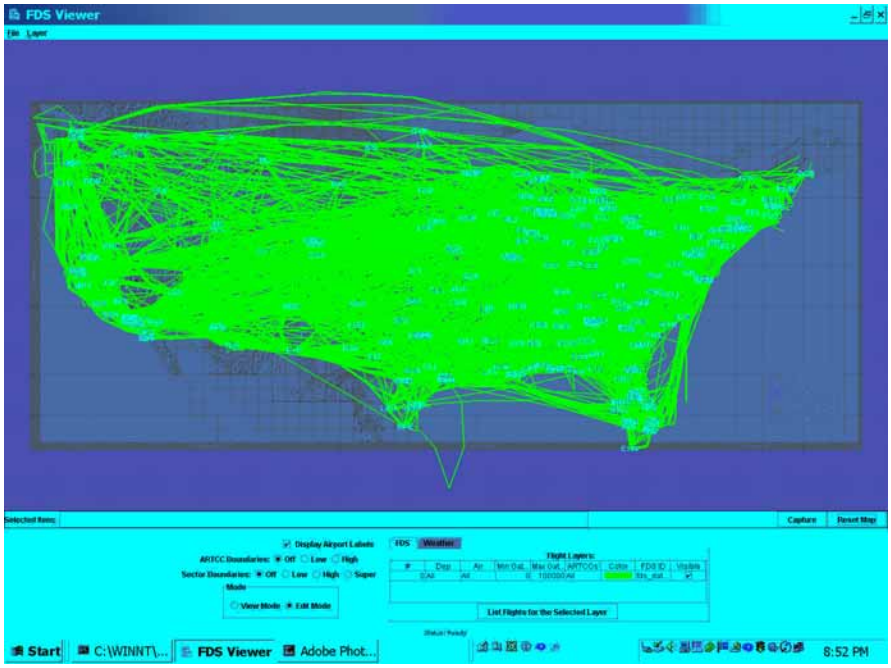

Figure 6: Visualization of the 5/17/02 Baseline Flight Data Set

4.2 Features

The major features of the Weather Editor / Flight Data Set Viewer include:

- Overall
  - 3D map GUI with full rotations/translation/zoom.
  - Clickable screen display.
- Weather Editing
  - Creation of an explicit representation of convective weather for ACES.
  - Ability to graphically draw areas affected by weather.
  - Ability to graphically draw the movement of storms as a series of weather affected areas, each with its own parameters.
- Weather Generation
  - Automated computation of sectors and airports affected by weather areas drawn by the user.
  - Computation of impacts on each affected airport and sector.

- Generation of scenario events corresponding to the affected airports and sectors.

✈ Weather Data Export

- Generation of ACES scenario data files from weather data.

✈ Flight Data Set Viewing

- Ability to visualize flight data sets in 3D (filed flight plans).
- Ability to display subsets of data based on airline/flightID, arrival and departure airports, time and ARTCCs traversed.

✈ Fully integrated with SPADES database and ACES Build 3.0.1.

4.3 Development Process

The Weather Editor is the first ACES data pre-processing tool prototypel. It is compliant with the SPADES concept and operates on a database rather than ACES text files. Because its functionality is new it was not built from pre-existing ACES components and in fact uses only a small number of ACES classes (primarily class *SharedData*). The program was, however developed to be consistent with ACES in that ACES coding standards, directory structures and CVS comment tags were used in all Java files. The Java package structure used for the software is also consistent with ACES and in fact the delivered SPADES code is designed to fit within the overall ACES directory hierarchy.

The software uses a Model-View-Controller software pattern which allows the user to make edits on various screens (in particular, on the map display and on a secondary spreadsheet window) while maintaining consistent data views across the screens and the underlying data structures.

4.4 Areas for Potential Future Enhancement

The Weather Editor demonstrates the SPADES pre-processing concept and is a fully functional tool. However, there are ways in which its utility and usability could be enhanced. Further, its connection to ACES is via scenario files as ACES does not have a Weather Agent. It is envisioned that the tool could be adapted to create data for a future explicit ACES weather representation as well.

Specific areas identified for potential enhancement include:

Item	**Discussion/Benefit**
Weather Editor	
1. Overlay real-world weather maps on weather editor display.	Facilitates drawing of Weather Affected Areas that closely mirror real weather.
2. Linkage to real-world sources	Investigate generating ACES weather data directly from real-world data

	sources.
3. Dynamic preview	Shows a preview of temporal evolution of flights (using ACES initialization data) and weather patterns. Useful for developing weather and flight data sets.
4. Integrate wind and convective weather	Currently Weather Affected Areas (WAAs) impact airport and sector capacities. This enhancement would override the default RUC values within WAAs, for example with a multiplier on wind speed or a local wind model.
Other Tools	
1. FDS generation tool	Combine SPADES database, FDS Viewer with NASA demand generation algorithms to create a demand generation / editing tool.

5 Evaluation of JView

5.1 Summary

The explicit goals of this task included producing "graphical displays that confirm the utility of the JView API for visualizing airspace simulation data" and "determine[ing] whether JView could form the basis for a replacement for the existing ACES visualization tool known as the VST." The tasks that were designed and performed as part of this project were specifically designed with these evaluations in mind.

During our work with JView we encountered both powerful strengths and hindering weaknesses within the tool. The sections below describe some of these experiences and suggest ways to improve the product.

As part of our evaluation of JView we also performed a survey of other similar graphics engines. In doing so we limited our scope to those packages that would be suitable for ACES: Java graphics engines that would operate across the likely range of supported ACES platforms (MS Windows, Linux and Mac OS X). This section also presents the results of that survey.

In general, the set of Java-based, multi-platform scene graph (scene graphs are discussed below) graphics engines is fairly sparse. There is no dominant package in terms of either features or market penetration. The JView API was created

because as of several years ago there was a true void in such a graphics capability and the JView engine remains a viable choice today.

The principal challenges in applying JView stem from the fact that it maintains the flavor of "research code" in its documentation, code quality and support/distribution mechanisms. The software has a strong set of features however some effort needs to be spent on improving the quality and usability of the code. Doing so would make JView a strong tool for Java-based simulation applications.

JView provides a rich visual environment for pre-processing tools. It support clickable, interactive graphics that allow users to, for example, drag and drop, draw and resize items on the screen. As such it is a good basis for pre-processing tools. One of the strengths of JView in this area, however is also in some ways a weakness. JView is a full 3D graphics engine and as such models a certain degree of 3D perspective. This can be visually confusing when viewing, for example, planned trajectories that are in fact on top of one another but which appear offset because they are being viewed from an angle. In much of the data entry for ACES the vertical dimension is separated from the horizontal dimensions (for example, a flight plan as a set of 2D waypoints vs. the flight's altitude profile) and so a 2D engine would be a useful tool addition to the JView toolkit.

A 2D engine would also be useful from a performance perspective. Because of the computational requirements of 3D graphics such an engine is always going to be slower than the simple raster graphics used in the VST. Thus, the answer of whether JView would be a viable substitute for VST is that it can display everything VST does and more, however running large numbers of flights with a more complex graphics engine is slower than running with VST. There is a performance price to be paid for higher end graphics. To some extent this performance cost could be mitigated by running JView-based applications on higher-end machines with GL optimized graphics, however such machines are not part of the typical ACES cluster. The runtime viewer application has not been tested on such hardware.

All that having been said, the JView engine remains a viable choice for 3D graphics visualization of ACES and NAS data, however in some instances a 2D engine more akin to a GIS would be a better match because of usability or performance reasons. The remainder of this section summarizes our specific findings relating to JView.

5.2 JView Strengths

1. No prior knowledge of OpenGL necessary

JView allows developers to create 3D applications with little to no prior knowledge of OpenGL or 3D graphics. To accommodate advanced users, the API offers access to the underlying, low-level OpenGL calls.

2. Scene graph architecture

JView uses a scene graph architecture which is generally considered very intuitive and well-suited for graphics development. A scene graph stores graphical elements in an acyclic tree in which the nodes represent elements, groups of elements, or transformations. For instance, a figure's "arm" node could contain a "hand" element which contains a group of "finger" elements. JView's scene graph implementation is logical and uncluttered, particularly when compared to lower level APIs such as Sun's Java3D.

3. Focus on performance

Java is often criticized for exhibiting poor performance when compared to programming languages such as C or C++. The JView team is very aware of these performance concerns and has proven very knowledgeable and conscientious when developing its product.

4. Developers are open to questions and code changes

The JView team has been very open to questions and comments. Their responses were quite clear and helpful. When our inquiries uncovered bugs or missing features, the team was willing to update the code accordingly.

5. API is available at no cost as GOTS

The JView API is developed and maintained by the Air Force Research Lab. It is available without cost to government users and programs. This is an advantage in environments where it is desirable to minimize total licensing costs.

5.3 JView Weaknesses

1. Documentation

The most notable problem we encountered with JView was its lack of complete, consistent documentation. It was difficult to get an overall picture of the functionality of the package. The "Introduction to JView" Word document did a good job of describing how scene element hits work, but omitted important information on scene element creation and oddments. The JavaDocs were also patchy with many uncommented parameters and methods. For instance, there was no mention that the TextImageContainer's selection feature was not functional, or that the TextRasterElement's font size/face could not yet be changed from the default values. In contrast, the DemoBrowser application contained sample code that was quite helpful. Among these three documents we were able to gather some idea of JView's capabilities and fill in the blanks through trial and error. Documentation improvements would greatly strengthen JView and make it accessible to a wider audience.

2. Incompleteness of Features

The Text classes mentioned above are but one example of cases where features in JView either weren't functional or didn't work as they should have. A number

of features, including the 2D API and “lenses”, turned out to be concepts rather than functioning code. These incomplete implementations were frustrating to work with, particularly given the afore-mentioned lack of documentation.

3. Communication problems

During development we often had difficulty consistently reaching the JView team by e-mail and telephone. At times we would receive multiple responses within a day while at other times we would have to wait 1-2 weeks for a response. One solution to this problem would be to implement a formal help or bug reporting system so users could be assured their inquiries have been received.

4. Formality in development

While concluding development of the Weather Editor, we found that certain properties of the Torus class had been removed in newer versions of JView. No path for backwards compatibility appeared to be available, so separate JView versions were included with the Weather Editor and ASRTV viewer. It is generally not clear which of the fixes AFRL provided will make it into the JView baseline and when they will do so. It would be helpful to users if these types of changes were formally considered and documented for groups using older releases.

5.4 Comparative Analysis

As mentioned above, as part of our evaluation of JView we surveyed similar similar graphics engines. We limited our scope to those packages that would be suitable for ACES: Java graphics engines that would operate across the likely range of supported ACES platforms (MS Windows, Linux and Mac OS X).

From our survey we conclude that scene graph APIs such as JView and Java3D are the most practical choices as they offer short development time and flexible feature sets. For comparison and completeness other categories of graphics APIs are also described.

1. Low-level graphics APIs

These APIs allow users to create graphics by accessing low-level OpenGL primitives. This is attractive because it grants the user complete control over visuals and performance. The downside to these products is that they require extensive graphics/mathematics expertise and generally increase development time when compared with scene graphs.

Products in this category include GL4Java (http://www.jausoft.com/gl4java.html), Java bindings for OpenGL (JOGL) (https://jogl.dev.java.net), and the Lightweight Java Game Library (LWJGL) (http://www.lwjgl.org/). They wrap OpenGL functions calls in a single class and are similar enough in syntax that porting code between them is easy. GL4Java

has not been updated since 2001 and is generally considered defunct. Sun is currently supporting JOGL as the standard OpenGL binding. LWJGL is a relatively new project that emphasizes speed for game development purposes. It should be noted that according to its documentation JView is based on GL4Java.

2. GIS-specific libraries

These libraries offer classes that assist with geographic visualization and coordinate manipulation. They generally emphasize 2D map displays with little support for 3D graphics. Many of them offer the capability to connect to a GIS database and create web-based client applications.

Products in this category include OpenMap, MapObjects, and ILOG JViews Maps.

OpenMap (http://openmap.bbn.com/) is an open source JavaBeans-based product created by BBN Technologies. It allows the user to pull information from databases to generate interactive maps, primarily using 2D graphics with some support for the Java3D API. BBN Technologies offers training, consulting services, and tech support contracts.

MapObjects (http://www.esri.com/software/mapobjects/) is a set of commercial 2D mapping components that support Java, Visual Basic, PowerBuilder, and Visual C++ development. It supports database connectivity, web connectivity, and a variety of CAD/GIS/image file formats.

ILOG JViews Maps (http://www.ilog.com/products/jviews/maps/) is a commercial Java-based library with an emphasis on light web clients. It offers capabilities to pull data from a variety of commercial map servers and allows developers to generate highly interactive 2D maps.

3. Scene graph-based Java APIs

Scene graph APIs (JView included) make use of acyclic tree structures to store a scene's graphical elements. The tree's objects, or nodes, usually consist of single elements or groups of elements. This structure is intuitive because it allows like items to be gathered in a logical way. The scene graph structure also allows a certain appearance or behavior to be applied to a group of nodes rather than having to iterate through each individual member. For instance, in the ASRTV viewer, all airport elements were grouped into a single node. This facilitated implementation of features such as toggling airport icons on and off.

In addition to JView, Java scene graph APIs include Java3D, Xith3D, and jME.

The Java3D API was developed by Sun from 1998 to 2002. There has been no significant development or releases as of late, but the API's source code was recently opened by Sun for community development. It is very well documented with detailed tutorials, JavaDocs, and manuals available online. Java3D has a

reputation for poor performance, however, this point is heavily contested by proponents. Unlike the other APIs described in this section, Java3D binds directly with OpenGL rather than using an intermediate layer such as JOGL or LWJGL. It also differs from the others because it limits access to low-level OpenGL calls. (http://java.sun.com/products/java-media/3D/)

Xith3D is an open source scene graph API that is built upon JOGL. It may also be used with LWJGL and offers access to low-level OpenGL calls through both binding sets. Xith3D emphasizes performance for use in game development. To achieve this goal, developers have left out features such as thread safety, which they feel greatly hinders performance in the Java3D API. It has limited documentation when compared to Java3D. (http://xith.org)

jME (jMonkey Engine) is another open source project similar to Xith3D. It was initially built on LWJGL with JOGL support expected in the near future. jME allows access to low-level OpenGL calls for advanced users. It shares Xith3D's goal of performance improvement for game creation and also drops thread safety to achieve this. It is well documented and has an active online community. (http://www.mojomonkeycoding.com)

6 Summary

The efforts performed under this project successfully developed and demonstrated a concept for ACES pre-processing tools. The JView graphics API was also evaluated and several prototypes were developed showing the applicability of this software to visualization of ACES data. All in all, graphical tools operating on a relational database such as were demonstrated in this project would provide a powerful data pre-processing environment for ACES. Such an environment would improve the simulation's usability. Application of analogous visualization tools to simulation outputs would similarly improve users' productivity in analyzing run results.

7 Acronyms

ACES	Airspace Concept Evaluation System
SPADES	Scenario Processing and Data Environment for Simulations
VAMS	Virtual Modeling and Simulation
VAST	Virtual Airspace Simulation Technologies
API	Application Programmer's Interface
SSDD	System/Subsystem Design Description
SDD	Software Design Document
NAS	National Airspace System
SCR	Software change Request
WAA	Weather Affected Area
AATT	Advanced Air Transportation Concepts and Technologies
SUA	Special Use Area

8 References

1. ACES System/Subsystem Design Description (SSDD). ATMSDI CTO7 CTOD 7.27
2. ACES Software Design Document (SDD). ATMSDI CTO7 CTOD 7.38.

This page intentionally left blank.